Q5： 最长回文子串。

1. 动态规划： boolean[][] dp = new boolean[n][n]; dp[i][j]: i~j是否是回文子串。

2. 中心扩散法： 每一个回文串的中心都是 一个字母，或者两个字母。我们枚举每一个这样的中心。对其进行扩展。 我们枚举所有的「回文中心」并尝试「扩展」，直到无法扩展为止，此时的回文串长度即为此「回文中心」下的最长回文串长度
长度为 1 和 2 的回文中心分别有 n 和 n-1 个

3. KMP : 模板匹配。s(j) : 作为p模式串，对s反转 s_rev(i):作为主串。

```
                    在kmp中while(i < s_rev.length())
                    则最后kmp将会匹配s_rev的最长后缀。
                    正常kmp 都是p较短，则当出现 while (i < ts.length() && j < ps.length())  j ==  ps.length()时，则说明已经找到了p串
                    而该题 我们缺失了 j < p.length()的条件，只有i越界时才会停止，所以匹配的是后缀。
                    当在找p前缀时，后缀时，由于i一直不停止，则即使在s中间找到了p串的前缀也不会停止匹配，而是继续向后匹配，
                    所以最后匹配的一定是 后缀，而不是中间的一部分。

                    当我们找到了p串的一个前缀是s串的后缀，那么该前缀就是一个回文串。然后我们将s串删除最后一个字符，p串从begin开始匹配。那么我们就能找到第二个前缀作为s串的后缀。然后两个串取最长。以此类推。
                    其实，这样找就是，找以 begin开头的回文串的最长长度。 每找一个前缀，begin后移一个。
```

Q10 : 正则表达式匹配

1. dp : 考虑*号的后效性，如从左向右匹配，那么*号到底吃多少个，是很难确定的。

```
 "aaa"
 "ab*a*c*a"  第二个a*，应该吃几个a是不确定的，因为后边还有一个a是必须吃一个，所以第二个a*吃几个a是无法确定的。
 所以，我们我们应该从右向左进行匹配，来消除这种后效性。

 dp[i][j] 表示 s[0~ i-1], p[0 ~ j-1] 是否匹配。  面向dp[i][j]是不考虑ij之后的内容的。
 dp[i][j]是一个独立的子问题。该子问题，只依赖于他自己的子问题 dp[<=i][<=j]
 dp[i][j]  依赖于之前的 dp[<=i][<=j]的结果。

 boolean table[][] = new boolean[s.length() + 1][p.length() + 1];
 table[0][0] = true : 空串总是能匹配空串
 s串为空串，p不为空时的匹配情况 : 要单独处理，仍然可能匹配成功。
```

Q31： 下一个排列

1. 654321： 是最大排列
2. 146532： 是 14 开头的最大排列， 此时递增只能用6532中大于4的最小元素5开始： 15开头
3. 而15开头的最小字典序为 15 2346 . 即交换大于4的最小元素5与4的位置，然后后边构成递减序列，逆置该递减序列，即为最小字典序。

Q43 . 字符串相乘： 给定两个以字符串形式表示的非负整数 num1 和 num2，返回 num1 和 num2 的乘积

```
            方法:    竖式。
                 12345
                   567
            ----------------------
                    35                用7 乘每一位，加在res的对应位置上 上
                   28
                  21
                 14
                 7
            ---------------------
                   30                 用 6
                  24
                 18
                12
                6
            --------------------
                  25
                 20
                15
               10
               5
            ----------------------
        res:
            每得到一个数，就可以确定res的一位，并向前一位进位，例如35，就可以确定5，并将3作
为进位提前复制给res,在得到28时，28+3 = 31 确定1，进位3.
```

Q131：分割回文串：对串分割，要求所有子串都是回文串，返回所有方案。

1. 首先，我们通过dp获得所有子串是否是一个回文串。dp[i][j] = true 则子串ij为回文串。
2. 通过回溯的 begin写法， 以当前位置为起点，准备分割下一个回文串。终点有很多选择。只要dp[i][j] = true即可。
3. 当cur == len时，说明已经分割完毕。存入list即可。

Q132：分割回文串 II： 求最少的分割次数。使得所有子串都是回文串。

1. 首先，我们通过dp获得所有子串是否是一个回文串。ishui[i][j] = true 则子串ij为回文串。
2. 对ishui再进行dp,得到，最小分割次数。dp[i]:到位置i时，的最少分割次数。

```
ishui[j<=i][i] = true 则dp[i] = max(dp[i], dp[j<i]+1) ;
```

Q205：同构字符串 s = "egg", t = "add" true s = "foo", t = "bar" false

方法1： 当来到位置 i时，s[i]的字符频率 应该等于 t[i]字符的频率。若从头开始迭代，就能递推出，两个串同构。

Q214： 最短回文串 s: "abcd" -> "dcbabcd" 字符串前加一段，使得该串变为回文串，要求回文串最短。

1. 实在不行就用动态规划得到最长回文子串，然后拼接。找最长前缀为回文串
2. 最好的方法就是： kmp 将s逆序，s_rev 在s_rev查找s, s的前缀恰好是s_rev的后缀。 当i走到头时，j恰好停在s串的前缀后一个位置

```
        next数组:
        四个 -1 , 两个 0，循环用while，不相等就回退 j = next[j]
        void getNext(String str, int[] next){
                int i = 0, j = -1; // j = -1 是特殊位置
                next[0] = -1;
                while(i < str.length() - 1){
                if(j == -1 || str.charAt(i) == str.charAt(j)){
                        i++;
                        j++;
                        next[i] = j;
                }else{
                        j = next[j]; // j = next[0] = -1; 表示回退到头了
                }
                }
        }

        kmp:
        int i = 0, j = 0;
        while(i < s_rev.length()){
                if(j == -1 || s_rev.charAt(i) == s.charAt(j)){
                        ++i;
                        ++j;
                }else{
                        j = next[j];
                }
        }
        正常kmp 都是p较短，则当出现 while (i < ts.length() && j <
ps.length())
        j ==  ps.length()时，则说明已经找到了p串
```

Q336：回文对

1. 在map中存所有串的逆序串
2. 对于串K，分为s1,s2, 若s1为回文串，那么在map中找s2若找到串t，那么串 t+s为一对回文串。 s1可能为空串，也可能是整个串。

Q1002 : 查找常用字符 ["bella","label","roller"] -> ["e","l","l"] e在每个单词中出现1次，l在每个单词中出现两次。

1. 不计空间的做法是，int[n][26] ,统计每个单词的字符频率。
2. int[26] min : min保存int[n][26]每一列的最小值。
3. 遍历 min[26]数组，就是ans.
4. 优化： int[n][26] -> int[26]:min的优化。 由于min数组保存的是int[n][26] 每一列的最小值。 我们只需要一个tempint[26]数组，计算几个单词，然后使用更新min的每一个频率即可。

字符串排序技巧

对于字符串的排序往往不用sort等算法， 而是基于字符频率的统计， 然后从小到大利用频率统计来拼字符串。时间可以达到 o(n),而只需要长度为26的int数组。 是很划算的。